/// leadController.ts

import { Request, Response, NextFunction } from 'express';

import leadService from '../services/leadService';

import { leadSchema } from '../validations/leadValidation';

export const createLead = async (req: Request, res: Response, next: NextFunction): Promise<void> => {

try {

const { error } = leadSchema.validate(req.body);

if (error) {

res.status(400).json({ error: error.details[0].message });

return; // Ensure all paths return a value or next()

}

const newLead = await leadService.createLead(req.body);

res.status(201).json({ message: 'Lead created successfully', data: newLead });

return; // Explicitly return here to satisfy TypeScript

} catch (error) {

next(error);

}

};

///leadModel.ts

import mongoose, { Document, Schema } from 'mongoose';

export interface ILead extends Document {

name: string;

phone: string;

email: string;

leadSource: string;

status: string;

}

const LeadSchema = new Schema<ILead>(

{

name: { type: String, required: true },

phone: { type: String },

email: { type: String,unique:true },

leadSource: { type: String, required: true },

status: { type: String, required: true, enum: ['New', 'In Progress', 'Closed'] }

},

{

timestamps: true // Enable timestamps

}

);

export default mongoose.model<ILead>('Lead', LeadSchema);

/// leadRepository.ts

import Lead, { ILead } from '../models/leadModel';

const createLead = async (leadData: ILead): Promise<ILead> => {

const lead = new Lead(leadData);

return await lead.save();

};

export default {

createLead

};

//leadService.ts

import leadRepository from '../repositories/leadRepository';

import Lead from '../models/leadModel'; // Import the Lead model

import { ILead } from '../models/leadModel';

const createLead = async (leadData: ILead) => {

// Check if a lead with the same email already exists

const existingLead = await Lead.findOne({ email: leadData.email });

if (existingLead) {

throw new Error('Email already in use. Please use a different email address.');

}

// Proceed to create the new lead

return await leadRepository.createLead(leadData);

};

export default {

createLead

};

//leadValidation.ts

// import Joi from 'joi';

// export const leadSchema = Joi.object({

// name: Joi.string().required(),

// phone: Joi.string().required(),

// email: Joi.string().email().required(),

// leadSource: Joi.string().required(),

// status: Joi.string().valid('New', 'In Progress', 'Closed').required()

// });

import Joi from 'joi';

export const leadSchema = Joi.object({

name: Joi.string().min(2).max(72).trim().required(),

phone: Joi.string().min(4).max(20).required(),

email: Joi.string()

.email()

.required()

.messages({

'string.email': 'Please enter a valid email address.',

'any.required': 'Email is required.',

'string.base': 'Email must be a string.',

}),

leadSource: Joi.string().required(),

status: Joi.string().valid('New', 'In Progress', 'Closed').required(),

});

// You can include a check for uniqueness in the service layer, as previously discussed.

//validateLead.ts

import { Request, Response, NextFunction } from 'express';

import { leadSchema } from './leadValidation';

const validateLead = (req: Request, res: Response, next: NextFunction): void => {

const { error } = leadSchema.validate(req.body);

if (error) {

res.status(400).json({ error: error.details[0].message });

return; // Ensure to return here to stop further execution

}

next(); // Or return next();

};

export default validateLead;

// lead/index.ts

import { Router } from 'express';

import { createLead } from './controllers/leadController';

import validateLead from './validations/validateLead';

const router = Router();

// post

router.post('/', validateLead, createLead);

// get

// router.get("/")

// // put

// router.put("/")

// // delete

// router.delete("/")

export default router;

// lead/types.ts

// src/APIs/lead/types.ts

export interface Lead {

name: string;

contactInfo: {

phone: string;

email: string;

};

leadSource: string;

status: string;

}  
  
  
**activity**

//activityModel.ts

import { Request, Response } from 'express';

import \* as activityService from '../services/activityServices'; // Adjusted import path

export const logActivity = async (req: Request, res: Response) => {

const { leadId, activityType, description, dateTime, assignedSalesRep } = req.body;

const newActivity = {

leadId,

activityType,

description,

dateTime,

assignedSalesRep,

};

try {

const activity = await activityService.createActivity(newActivity);

return res.status(201).json(activity); // Return the response

} catch (error) {

return res.status(400).json({ error: 'Failed to log activity.' }); // Return the response

}

};

export const getActivityById = async (req: Request, res: Response) => {

const { id } = req.params;

try {

const activity = await activityService.getActivityById(id);

if (!activity) {

return res.status(404).json({ error: 'Activity not found.' });

}

return res.status(200).json(activity); // Return the response

} catch (error) {

return res.status(400).json({ error: 'Failed to retrieve activity.' }); // Return the response

}

};

export const getAllActivities = async (\_: Request, res: Response) => {

try {

const activities = await activityService.getAllActivities();

return res.status(200).json({ data: activities }); // Return the response

} catch (error) {

return res.status(400).json({ error: 'Failed to retrieve activities.' }); // Return the response

}

};

export const updateActivity = async (req: Request, res: Response) => {

const { id } = req.params;

const updateData = req.body;

try {

const updatedActivity = await activityService.updateActivity(id, updateData);

if (!updatedActivity) {

return res.status(404).json({ error: 'Activity not found.' });

}

return res.status(200).json(updatedActivity); // Return the response

} catch (error) {

return res.status(400).json({ error: 'Failed to update activity.' }); // Return the response

}

};

export const deleteActivity = async (req: Request, res: Response) => {

const { id } = req.params;

try {

const deletedActivity = await activityService.deleteActivity(id);

if (!deletedActivity) {

return res.status(404).json({ error: 'Activity not found.' });

}

return res.status(200).json({message:"Activity deleted successfully"}); // No content response

} catch (error) {

return res.status(400).json({ error: 'Failed to delete activity.' }); // Return the response

}

};

//activityModel.ts

// /src/features/activities/models/activityModel.ts

import mongoose, { Document, Schema } from 'mongoose';

export interface IActivity extends Document {

leadId: mongoose.Types.ObjectId; // Reference to the Lead

activityType: 'Call' | 'Email' | 'Meeting';

description: string;

dateTime: Date;

assignedSalesRep: string; // Name or ID of the sales representative

}

const activitySchema: Schema<IActivity> = new Schema({

leadId: { type: Schema.Types.ObjectId, required: true, ref: 'Lead' },

activityType: { type: String, enum: ['Call', 'Email', 'Meeting'], required: true },

description: { type: String, required: true },

dateTime: { type: Date, default: Date.now },

assignedSalesRep: { type: String, required: true },

});

export default mongoose.model<IActivity>('Activity', activitySchema);

//activityRepository.ts

// /src/features/activities/repositories/activityRepository.ts

import Activity from '../models/activityModel';

import { IActivity } from '../models/activityModel';

export const createActivity = async (activityData: IActivity): Promise<IActivity> => {

const activity = new Activity(activityData);

return await activity.save();

};

export const getActivityById = async (id: string): Promise<IActivity | null> => {

return await Activity.findById(id).populate('leadId'); // Populate to get lead details if necessary

};

export const getAllActivities = async (): Promise<IActivity[]> => {

return await Activity.find().populate('leadId');

};

export const updateActivity = async (id: string, updateData: Partial<IActivity>): Promise<IActivity | null> => {

return await Activity.findByIdAndUpdate(id, updateData, { new: true }).populate('leadId');

};

export const deleteActivity = async (id: string): Promise<IActivity | null> => {

return await Activity.findByIdAndDelete(id);

};

//activityServices.ts

import Activity from '../models/activityModel';

// Create a new activity

export const createActivity = async (activityData: any) => {

const activity = new Activity(activityData);

return await activity.save();

};

// Get activity by ID

export const getActivityById = async (id: string) => {

return await Activity.findById(id); // Assuming you are using Mongoose

};

// Get all activities

export const getAllActivities = async () => {

return await Activity.find(); // Retrieves all activities

};

// Update activity by ID

export const updateActivity = async (id: string, updateData: any) => {

return await Activity.findByIdAndUpdate(id, updateData, { new: true }); // Updates and returns the updated document

};

// Delete activity by ID

export const deleteActivity = async (id: string) => {

return await Activity.findByIdAndDelete(id); // Deletes the activity and returns it

};

// You can add other service methods as needed

//activityValidation.ts

// /src/features/activities/validations/activityValidation.ts

import Joi from 'joi';

export const activitySchema = Joi.object({

leadId: Joi.string().required(),

activityType: Joi.string().valid('Call', 'Email', 'Meeting').required(),

description: Joi.string().required(),

dateTime: Joi.date().iso().default(Date.now),

assignedSalesRep: Joi.string().required(),

});

// validate.ts

// /src/middlewares/validationMiddleware.ts

import { Request, Response, NextFunction } from 'express';

import { ObjectSchema } from 'joi';

const validate = (schema: ObjectSchema) => {

return (req: Request, res: Response, next: NextFunction) => {

const { error } = schema.validate(req.body, { abortEarly: false }); // Validate request body against the schema

if (error) {

return res.status(400).json({

errors: error.details.map(err => ({

message: err.message,

path: err.path,

}))

});

}

return next(); // Explicitly return next()

};

};

export default validate;

//activities/index.ts

// /src/features/activities/index.ts

import { Router } from 'express';

import {

logActivity,

getActivityById,

getAllActivities,

updateActivity,

deleteActivity,

} from './controllers/activityController';

import { activitySchema } from './validations/activityValidation';

import validate from './validations/validate';

const router = Router();

// Create Activity

router.post('/', validate(activitySchema), logActivity);

// Get All Activities

router.get('/', getAllActivities);

// Get Activity by ID

router.get('/:id', getActivityById);

// Update Activity

router.patch('/:id', validate(activitySchema), updateActivity);

// Delete Activity

router.delete('/:id', deleteActivity);

export default router;

//activities/types.ts

// /src/features/activities/types.ts

import { Types } from 'mongoose'; // Import mongoose Types

import { IActivity } from './models/activityModel'; // Adjust the import path if necessary

export type ActivityType = 'Call' | 'Email' | 'Meeting';

export interface Activity extends IActivity {

leadId: Types.ObjectId; // Use ObjectId type to match the IActivity interface

activityType: ActivityType;

description: string;

dateTime: Date;

assignedSalesRep: string;

}

export interface CreateActivityInput {

leadId: Types.ObjectId; // Match the type with IActivity

activityType: ActivityType;

description: string;

dateTime?: Date; // Optional, as it can default to now

assignedSalesRep: string;

}

**Opportunity**//opportunitController.ts

import { Request, Response, NextFunction } from 'express';

import opportunityService from '../services/opportunityService';

import { opportunitySchema } from '../validations/opportunityValidation';

export const createOpportunity = async (req: Request, res: Response, next: NextFunction): Promise<Response | void> => {

try {

const { error } = opportunitySchema.validate(req.body);

if (error) {

return res.status(400).json({ error: error.details[0].message });

}

const opportunity = await opportunityService.createOpportunity(req.body);

return res.status(201).json({ message: 'Opportunity created successfully', data: opportunity });

} catch (error) {

return next(error); // Pass the error to the next middleware

}

};

//opportunityModels.ts

import mongoose, { Document, Schema } from 'mongoose';

export interface IOpportunity extends Document {

leadId: string;

projectName: string;

expectedCloseDate: Date;

projectValue: number;

stage: string;

createdAt?: Date;

updatedAt?: Date;

}

const OpportunitySchema = new Schema<IOpportunity>({

leadId: { type: String, required: true },

projectName: { type: String, required: true },

expectedCloseDate: { type: Date, required: true },

projectValue: { type: Number, required: true },

stage: { type: String, required: true, enum: ['Bid', 'Negotiation', 'Closed', 'Lost'] },

}, { timestamps: true });

export default mongoose.model<IOpportunity>('Opportunity', OpportunitySchema);

//opportunityService.ts

import Opportunity from '../models/opportunityModel';

import { IOpportunity } from '../models/opportunityModel';

const createOpportunity = async (opportunityData: IOpportunity) => {

const opportunity = new Opportunity(opportunityData);

return await opportunity.save();

};

export default {

createOpportunity

};

//opportunityValidation.ts

import Joi from 'joi';

export const opportunitySchema = Joi.object({

leadId: Joi.string().required(),

projectName: Joi.string().required(),

expectedCloseDate: Joi.date().required(),

projectValue: Joi.number().positive().required(),

stage: Joi.string().valid('Bid', 'Negotiation', 'Closed', 'Lost').required(),

});